AMENDMENTS TO THE SPECIFICATION

Please replace the paragraph beginning on page 1, line 4, with the following rewritten paragraph:

-- This application claims priority to U.S. Provisional Patent Application No. 60/151,531 entitled "SYSTEM AND METHOD FOR PROVIDING COMPUTER SECURITY" filed August 30, 1999, which is incorporated herein by reference for all purposes. ~~, and to~~ U.S. Patent Application No. 09/615,697 entitled "SYSTEM AND METHOD FOR COMPUTER SECURITY" filed July 14, 2000, ~~which~~ is incorporated herein by reference for all purposes. --

Please replace the paragraph beginning on page 1, line 10, with the following rewritten paragraph:

--This application is related to co-pending U.S. Patent Application No. 09/651,439 ~~(Attorney Docket No. RECOP011)~~ entitled SYSTEM AND METHOD FOR DETECTING COMPUTER INTRUSIONS filed concurrently herewith, which is incorporated herein by reference for all purposes; and co-pending U.S. Patent Application No. 09/651,303 ~~(Attorney Docket No. RECOP012)~~ entitled EXTENSIBLE INTRUSION DETECTION SYSTEM filed concurrently herewith, which is incorporated herein by reference for all purposes; and co-pending U.S. Patent Application No. 09/651,434 ~~(Attorney Docket No. RECOP014)~~ entitled SYSTEM AND METHOD FOR USING SIGNATURES TO DETECT COMPUTER INTRUSIONS filed concurrently herewith, which is incorporated herein by reference for all purposes; and co-pending U.S. Patent Application No. 09/651,304 ~~(Attorney Docket No. RECOP015)~~ entitled SYSTEM AND METHOD FOR ANALYZING FILESYSTEMS TO DETECT INTRUSIONS filed concurrently herewith, which is incorporated herein by reference for all purposes; and co-pending U.S. Patent Application No. 09/651,306 ~~(Attorney Docket No. RECOP016)~~ entitled SYSTEM AND METHOD FOR DETECTING BUFFER OVERFLOW ATTACKS filed

concurrently herewith, which is incorporated herein by reference for all purposes; and co-pending U.S. Patent Application No. 09/651,347 ~~(Attorney Docket No. RECOP017)~~ entitled SYSTEM AND METHOD FOR USING TIMESTAMPS TO DETECT ATTACKS filed concurrently herewith, which is incorporated herein by reference for all purposes.--

Please replace the paragraph beginning on page 16, line 7, with the following rewritten paragraph:

– The system shown in Figure 2 also includes a trap system 210, comprising comprises a trap host system 212 in which a virtual cage 214 is established, as described in co-pending U.S. Patent Application No. ~~09/615,697~~ 09/615,967. Trap system 210 also includes an administration console 216 connected to trap host system 212 and configured to enable a system administrator (or other authorized user) to control the configuration of trap host system 212 and virtual cage 214. Trap system 210 also includes a database 218 used to store data relating to activities within trap host system 212 and virtual cage 214. –

Please replace the paragraph beginning on page 24, line 13, with the following rewritten paragraph:

– The inventive system may also search the filesystem, including deleted entries, for filenames and filename patterns that are known parts of attacks, such as names that are part of attack scripts in circulation or use, and names that are part of the standard operating practice/modus operandi of attackers. Filesystem information, both timestamps and file signatures, may be recovered from backup dumps without having to reload the files and directories to disk. In an embodiment of the invention, the system supports the ufsdump format, which is the most commonly used on a range of UNIX® systems, and supports additional dump formats with data collection modules as needed. –

Please replace the paragraph beginning on page 31, line 16, with the following rewritten paragraph:

– The system of the invention may be configured to operate with various computing platforms, singly and in combination. However, similar data sources on related platforms have

small but critical variability, such as different subsets of the data fields and different data representations. For example, the UNIX® uid_t (user id) data-type may change from a 16-bit integer to a 32-bit integer across platforms. On some platforms, it is a signed value, and on others, it is an unsigned value (i.e., non-negative). Some hardware architectures are little-endian (e.g., Intel x86), while others are big-endian (e.g., SPARC®). Some use 32-bit words and others use 64-bit words. –

Please replace the paragraph beginning on page 37, line 21, with the following rewritten paragraph:

– Most semantic types can be treated as distinct items, that is, the semantic type is a single feature, not a set of features. The primary exceptions are semantic types that involve time. Different platforms use different encodings of time. For example, UNIX® platforms keep time as the number of seconds from 1970-01-01 00:00:00 UTC (Universal Coordinated Time), while MacOS uses 1904-01-01 00:00:00 UTC. Semantic types that are a combination of features are assigned integer values where bit fields are allocated to the different features, allow the algorithms to exploit these patterns. –

Please replace the paragraph beginning on page 38, line 14, with the following rewritten paragraph:

– A related issue with time reports is that of granularity. In UNIX®, the default granularity is seconds (time_t), but some log files record time in human-readable form at a granularity of only minutes, and some events are recorded with higher precision by using a structure in which the first element is in seconds (time_t) and the second element encodes the subinterval, either microseconds (struct timeval) or nanoseconds (struct timespec). This granularity is encoded into the semantic type as a bit-field, paralleling the encoding of the time-origin. –

Please replace the paragraph beginning on page 39, line 1, with the following rewritten paragraph:

– The next level of abstraction in the meta-protocol is the message, which is composed of a header and an unordered collection of data-items. Different platforms have different sets of values, for example, the UNIX® filesystem records three time values for each file: last-access

time, last-modification time, and last-change time (where change is traditionally defined to be either a modification to the file's contents or a change to its properties). Other types of filesystems record subsets of these, such as the last-modification time only. Sensors report only the data that they can extract, and do not send values encoding unavailable, nor do they try to extrapolate values. Because the analysis engine looks for subtle inconsistencies, extrapolation carries substantial risk of misleading the analysis process. Distinguished values for unavailable are often not practical, because the designers of the platforms where that data is available typically did not reserve any values for this purpose, and even where there are reserved or unused values that can be usurped for this purpose, it is highly unlikely that the same value will be available across all platforms where it is needed. Unavailable/undefined values are assigned by the analysis engine based on the features of the database being used. –

Please replace the paragraph beginning on page 43, line 17, with the following rewritten paragraph:

– For UNIX® and its variants, the init (process control initialization: the parent of all other processes) creates a getty process for all lines on which logins are to be enabled. This includes both physical connections (console, terminal lines, modems, etc.) and network connections. getty initializes the line and monitors for a connection attempt, at which point it invokes a login process. If the user successfully logs in, the login process exec's the specified shell for the user (exec replaces the program running as the current process with a new program, as opposed to running the new program as a child process of the current process). A failed login attempt or the end of a successful login session generates a signal to the getty that triggers it to re-initialize the line and await the next login attempt. A failed login attempt occurs when the user has failed to enter a valid username-password pair within the allotted interval or has exceeded the allotted number of attempts to enter a valid pair. –

Please replace the paragraph beginning on page 44, line 1, with the following rewritten paragraph:

– Sometimes a user switches from one account into another account to execute a few commands before returning to the original account. The most common use of this is for a system administrator to switch from his normal (unprivileged) user account to the *root* (superuser) account to perform a few privileged operations (*e.g.*, system administration, software

installation) and then return to unprivileged status. Other common usages involve users temporarily switching from their personal accounts to a functional account (*e.g.*, application or project administrator) or to a group account. Having to logout and log back in would be too inconvenient (and slow) and would encourage users to subvert the reasons for having separate accounts. To avoid this situation, the *su* command (Substitute User) allows a user to easily switch between accounts. Logging of *su*'s has some minor variation over platforms. For example, in Solaris®, reports go to the log file *sulog*, while in Linux, the reports use the *syslog* system and are sent to its *authentication* facility. –

Please replace the paragraph beginning on page 44, line 8, with the following rewritten paragraph:

– The recording of the login process has minor variations over the variants of UNIX®. The stereotypical pattern is that when a valid username-password pair is entered, the login process writes a record to the utmp and wtmp files and updates the lastlog file. The utmp file tracks who is currently logged in, and the wtmp file provides a historical record, including both completed login sessions and active sessions. The lastlog file contains the time of the last login for each user, and the previous value is written to the user's terminal as part of the "hello" message. When the user logs out, the getty process removes the corresponding entry from the utmp file and writes a session-end record to the wtmp file. The getty process must perform this task because the login program is no longer present (it replaced itself with the user's shell program), and the user's shell cannot be trusted to make these updates: the shell may terminate abnormally (i.e., not have a chance to do the update), or the author of the shell program may forget to do this (users can create custom shells). –

Please replace the paragraph beginning on page 44, line 21, with the following rewritten paragraph:

– The details of recording of failed logins varies over platforms. Most platforms write reports of failed logins to the *authentication* facility of *syslog*, and some write to a designated file (*e.g., loginlog* in Solaris). For most, the threshold for reporting is, by definition, the maximum number of attempts allowed before the connection is severed. Consequently, most modern password-guessing attacks involve a single guess per connection, thereby not generating any *explicit* reports of a failed login attempt. –

Please replace the paragraph beginning on page 46, line 14, with the following rewritten paragraph:

– Network services that provide terminal-like interactions (*e.g., telnet*, *rlogin*, and *ftp*) use *pseudo-terminals* to emulate the drivers for hard-wired terminals. When a connection is made to one of these services, a pseudo-terminal is allocated and the server writes a record to *wtmp*, but this is simply a convention, not an enforced requirement. Services that use the *login* program (*e.g., telnet, rlogin*) have records written to *utmp* and *wtmp* the same as hard-wired lines. However, some servers such as FTP allow access similar to login, but by a separate mechanism. Some of these record these "logins" in *utmp* and *wtmp* and some do not. For example, the Solaris® FTP daemon does not, but the WUSTL (Washington University in St. Louis) FTP daemon does. –

Please replace the paragraph beginning on page 49, line 5, with the following rewritten paragraph:

– Files with holes. In UNIX® and its variants, files are composed of a sequence of 512-byte disk blocks, but these blocks do not need to be continuous on the disk, or even in the same relative order. The i-node for the file contains an ordered list of the addresses of the blocks that contain the contents of the file. If all the bytes in one of these blocks have the value zero, the block does not need to be allocated, and its address is instead given as zero. This significantly reduces the space used by certain types of files, typically binary executable files where there are large global data structures that are initialized to zero. But it also occurs in other binary files, such as lastlog. –

Please replace the paragraph beginning on page 52, line 4, with the following rewritten paragraph:

– In an embodiment of the invention, the ~~system~~ analysis engine 802 collects data related to logins to the host 804 with multiple sensors, such as:

a) the Directory-Tree Scanner 806 that collects information from the directories and from the *i-nodes*

b) the sensor for the password file 808 (and shadow password file if it exists)

c) sensors 810 for each of the logfile formats:

i) *cron* 812 and *at* 814 logs

ii) *lastlog* 816

iii) *sulog* 818

iv) *syslog* 820

v) *utmp* 822 / *wtmp* 824 –

Please replace the paragraph beginning on page 52, line 15, with the following rewritten paragraph:

– *Configuration discovery:* Except for *syslog*, these log files have standard locations, with some variance between platforms. For example, *lastlog* is in directory */var/adm* on Solaris® and in directory */var/log* on Linux. The pathnames for the *syslog* files are extracted by a data collection sensor from the *syslog.conf* file. –

Please replace the paragraph beginning on page 54, line 3, with the following rewritten paragraph:

– Different platforms have vastly different sizes of *struct lastlog*. On 32-bit Solaris®, it is 292 bytes, or more than half of the disk block. Thus, a block containing all nulls will implicate at most three consecutive user accounts. However, on Linux 2.2, the size is only 28 bytes, and thus there is a range of 20 User IDs implicated. –

Please replace the paragraph beginning on page 61, line 5, with the following rewritten paragraph:

– At the core of the UNIX® File System is the i-node, which contains the file's properties (e.g., owner, permissions) and pointers to the sequence of disk block containing the file's contents. An i-node does not contain the name of the file, thereby allowing files to have multiple names (hard links). –

Please replace the paragraph beginning on page 62, line 13, with the following rewritten paragraph:

– In the UNIX® filesystem, files are not directly deleted. Instead, they are unlinked from directories; i.e., the mapping from the filename to the i-node is deleted. When the number of links drops to zero, the i-node is deleted. Since virtually all files have a only single name, "unlinking a filename" is commonly referred to as "deleting a file." When a filename is unlinked, the bytes used by its dirent are added to the unused bytes after the filename in the immediate preceding dirent, and the i-node value is set to zero. –

Please replace the paragraph beginning on page 65, line 2, with the following rewritten paragraph:

– In one embodiment, the analysis engine starts at the beginning of the directory, stepping through the active dirents. In a UNIX® filesystem, the "." and ".." must be present for the directory to be valid. This provides a simple initial condition for the iteration. –

Please replace the paragraph beginning on page 66, line 9, with the following rewritten paragraph:

– (b) The filename does not contain any illegal characters. In the UNIX® file system, the illegal characters are the string terminator (0) and the character slash ('/'). –

Please replace the paragraph beginning on page 69, line 6, with the following rewritten paragraph:

– The Berkeley Fast File system is the basis of the native filesystems on most variants of UNIX®. To improve locality of files and avoid the need to periodically de-fragment the disk, it subdivides disk partitions into cylinder groups (typically 16 cylinders per group). Each cylinder group has its own set of i-nodes and data blocks. Its placement algorithm for a new file is to use an i-node in the same cylinder group as the directory entry it is linked to. The initial data blocks

for the file also go in the same cylinder group, but very large files have their data blocks spread over multiple cylinder groups to avoid them taking a disproportionate share from any one cylinder group. Further details of this placement will be apparent to one skilled in the art. –

Please replace the paragraph beginning on page 69, line 18, with the following rewritten paragraph:

– The file system occasionally gets corrupted, either from a hardware fault or because the system failed to complete a sequence of write operations. UNIX® has historically provided utilities that provided varying levels of help in repairing various levels of damage to the disk. These tools can work reasonably well for smaller files, but have significant limitations for larger files. There are third party tools that reverse the disk block allocation algorithm to improve the accuracy of disk blocks used to re-constitute a file. –

Please replace the paragraph beginning on page 77, line 1, with the following rewritten paragraph:

– Currently, the most common exploits involve a buffer overflow attacks on SetUID commands. A SetUID (also "SUID") command is one that runs with the privileges of the owner of the command instead of with the privileges of the user invoking the commands, and this attribute is specified by a flag in the permissions for the command (an executable file). The ownership of files and processes in UNIX® (and variants) is specified by an integer called the UID (User Identifier). Thus, the name SetUID comes from the operating system setting the UID of the command's process to be that of the owner of the file. –

Please replace the paragraph beginning on page 78, line 8, with the following rewritten paragraph:

– In UNIX® and its variants, most SetUID commands run with root (superuser) privilege, and the typical buffer overflow exploit for these commands is to have them give the user a shell running with root privilege, thereby allowing them unlimited access to the host. These attacks are sensitive to the exact formation of the data used to overflow the buffer. –

Please replace the paragraph beginning on page 78, line 18, with the following rewritten paragraph:

– The inventive system includes a database of SetUID commands and the files they access (in UNIX®, the file system is the interface to system resources), and how (read, modify, etc.). The analysis engine examines the last-access time of each SetUID command -- this is a reasonable approximation of when the command was last run, because there are a few other operations that update the last-access time, but these are uncommon/infrequent (e.g., making a copy of the executable, searching the executable for strings and symbols). This access time is compared to the timestamps on files that the command is expected to access. If those timestamps are earlier than the last-access time on the SetUID command, this is evidence that a SetUID buffer overflow attack may have occurred. For example, the eject and fdformat commands in Solaris® 2.5 are vulnerable to this attack. The eject command cause removable disks (floppies, CDs, etc.) to be ejected from the drive. A legitimate user may issue an eject command to check if any media is in the drive, leading to a false positive. A false positive can also arise if the user executes the eject command in a window connected to a host other than the one intended. The fdformat command formats floppy disks. While it wouldn't be usual for someone to execute fdformat in the wrong window (as with eject), it would be very unusual for him to execute it if there wasn't a floppy disk in the drive. –